Intra-Epoch Message Scheduling To Exploit Unused or Residual Overlapping Potential

Judicael A. Zounmevo
ECE Dept., Queen's University
19 Union Street, Kingston, ON, Canada K7L 3N6
judicael.zounmevo@queensu.ca

Ahmad Afsahi
ECE Dept., Queen's University
19 Union Street, Kingston, ON, Canada K7L 3N6
ahmad.afsahi@queensu.ca

ABSTRACT

As part of a new MPI Remote Memory Access (RMA) implementation over remote direct memory access, we propose a message scheduling scheme that interleaves inter-node and intra-node data transfers in a way that minimizes the overall latency of the RMA epoch. By doing so, we fulfill inter-node communication/intra-node communication overlapping and show that further latency mitigation is possible when computation length and communication payload sizes are too unbalanced to allow a satisfying level of communication/-computation overlapping. Test results show that the proposed scheduling scheme compares favorably against the approaches used in MVAPICH and Open MPI.

Keywords

MPI, RMA, One-sided, Message scheduling, Overlapping

1. INTRODUCTION

In order to achieve minimal latency and keep the CPU available for other duties when communication-related data transfer is being fulfilled, nowadays supercomputers are equipped with network devices featuring a remote direct memory access (RDMA) engine. However, it is common and even recommended, to perform intra-node data transfers using shared-memory; that is, RDMA does not intervene in communications that do not cross node boundaries. In fact, unless, I/O acceleration technologies such as Intel I/OAT [4] is available, intra-node data transfers keep the CPU busy in most supercomputers. Thus, the rest of the analysis in this work focuses on the more common situation of absence of I/O acceleration technologies.

The one-sided communication model of the Message Passing Interface (MPI) [7] is based on the concept of epoch. An epoch is a region enclosed by a pair of matching opening and closing synchronizations. Inside the epoch, the one-sided communications, also called RMA, are always nonblocking. The epoch-closing synchronization is blocking and does not exit until all the communications hosted by the epoch are completed, at least locally. The split-phase tandem created by the RMA communications and the blocking synchronizations creates the adequate conditions for communication/computation overlapping. Communication/computation overlapping is a widespread communication latency mitigation mechanism in high performance computing. The mechanism parallelizes data transfer and computation; and by doing so, it prevents as much as possible any of the CPU or the network device from idling.

When there is no computation to overlap to the data transfer occurring inside the epoch, for instance because of data dependency, communication/computation overlapping opportunities cannot be leveraged. However, as mentioned previously, the absence of computation is not synonymous with CPU inactivity, especially if the epoch contains intra-node RMA communication. In this work, we fully exploit the overlapping potential offered by the possible activity of the two *engines* embodied by RDMA and the CPU. In absence of computation inside the RMA epoch we overlap the intra-node communication driven by the CPU with the data transfer time of the RDMA engine. When computation does exist inside the epoch but is insufficient to fully exploit the overlapping potential offered by the inter-node data transfer time, we overlap the unused residual overlapping opportunity with intra-node communication as well. We perform those intra-node/inter-node communication overlapping with a message scheduling scheme that we apply to the RMA communications inside each epoch. We show that the message scheduling, and the resulting intra-node/inter-node communication overlapping reduces the overall latency of the epoch. To the best of our knowledge, this is the first study on message scheduling in MPI one-sided communication in order to reduce overall epoch latency.

The rest of the paper is organized as follows. Section 2 describes the MPI one-sided communication model. Section 3 discusses the motivation for this work. Section 4 explains the design. Section 5 shows experimental results. Section 6 covers related work and Section 7 concludes the paper.

EuroMPI/ASIA '14, September 9-12 2014, Kyoto, Japan
Copyright 2014 ACM 978-1-4503-2875-3/14/09 ...$15.00.
http://dx.doi.org/10.1145/2642769.2642779.

2. BACKGROUND

In one-sided communications between any two peers, a single process, the *origin*, specifies all the communication parameters while the other one, the *target*, remains passive. The communication occurs over objects called *windows* which act as communication scope and define the memory regions that each process intends to expose for remote access. MPI RMA occurs in critical section-like regions called *epochs*. An epoch is defined by a pair of opening and closing synchronization constructs. When an epoch is over, all its contained RMA communications are guaranteed to have completed for the concerned process. MPI defines two classes of RMA synchronizations, namely *active target* where the target explicitly opens an epoch to match origin-side epochs; and *passive target* where the target does not need to make epoch calls. There are two kinds of active target epochs: *Fence* and General Active Target Synchronization (GATS). In a fence epoch, all processes of the RMA window are potentially both origin and target. In a GATS epoch, a process is either exclusively origin or exclusively target. An origin can open an access epoch over any subset of the processes encompassed by the RMA window. Similarly, a target can open an exposure epoch over any subset of the RMA window. In a passive target epoch, an origin process locks either a specific target or all the processes encompassed by the RMA window. If the passive target epoch is towards a single target, the lock can be requested either exclusively, in which case it is held by a single process at a time; or in a shared fashion, in which case it can be acquired even if other processes already hold it in a shared fashion as well. A passive target epoch opened towards all the processes of the RMA window can only request shared locks.

RMA communications can load from (e.g., `MPI_GET`), store to (e.g., `MPI_PUT`), or update the memory attached to a remote window (e.g., `MPI_ACCUMULATE`, `MPI_GET_ACCUMULATE`). The updates can bear an arbitrarily large payloads made of an array of a given datatype. In the special cases where the payload is a single instance of certain predefined datatypes, MPI offers `MPI_FETCH_AND_OP` as a faster lightweight alternative to `MPI_ACCUMULATE`. `MPI_COMPARE_AND_SWAP` is also defined in the family of update routines. MPI RMA communications are guaranteed to be completed only after the synchronization call that closes the hosting epoch exits. In passive target, that constraint can be relaxed either with the `flush` family of routines (e.g., `MPI_WIN_FLUSH`) or with request-based RMA routines.

3. MOTIVATION

Radix [17] is a non-comparative sort approach that can be faster than $O(nlog(n))$ approaches such as Quicksort. Radix sort can be particularly interesting when it comes to sorting integers or fixed-length strings of any finitely enumerable type. We designed a one-sided implementation of Radix that overlaps computation with communication as much as possible. However, with sufficiently large datasets, the execution becomes considerably dominated by communication time; we have experienced ratios of communication to computation of close to 100. Moreover, with only 8 CPU cores per node on our test systems, each process tends to perform substantially more inter-node communication than intra-node communication. In summary, the algorithm offers a tremendous potential for overlapping due to the amount of inter-node communication; but there is not enough computation to exploit the overlapping potential. This observation encourages our message scheduling scheme meant to consume the substantial residual overlapping potential with intra-node communication.

As a reminder, in the absence of I/O acceleration technologies, intra-node communications which are shared-memory-based, use the CPU for the whole duration of their data transfers. In order to analyze the overlapping possibilities, we define α_I, D_I and ε_I, respectively as the CPU time needed to submit data transfers to the RDMA engine for inter-node communications, the time needed for the RDMA engine to stream the data, and the time needed by the CPU to detect all completions from the RDMA engine activity. We define ζ_i as the CPU time needed to do all the intra-node memory copies. We also define C_p as the computation inserted inside the epoch for communication/computation overlapping purpose. In order to remove from the analysis the delay caused by origins deferring their RMA transfers because of late targets, we assume that the exposure epoch (if relevant), or the RMA lock (if relevant), is granted before the RMA communications are issued. The minimal possible epoch latency is:

$$L_0 = \alpha_I + \varepsilon_I + max(D_I, \zeta_i + C_p) \tag{1}$$

An epoch can be divided in three sections that are 1) the opening (e.g., `MPI_WIN_START`), 2) the intra-epoch, and 3) the epoch closing (e.g., `MPI_WIN_COMPLETE`). The intra-epoch, which is located between the opening and the closing, is the only section where communication/computation overlapping can occur. Thus it is customary in MPI implementations built over RDMA to trigger all large-payload inter-node RMA right inside the intra-epoch section if possible. Small-payload inter-node communications can be deferred for various reasons (e.g., message aggregation). Deferring small-payload communications is also justified because they bear insignificant overlapping potential. As for intra-node communications, they are also deferred to the closing phase of the epoch because they leave no room for communication/computation overlapping. The epoch latency in this case is thus expressed by:

$$L_1 = \alpha_I + \varepsilon_I + max(D_I, C_p) + \zeta_i \tag{2}$$

One can notice that:

$$max(D_I, \zeta_i + C_p) \leq max(D_I, C_p) + \zeta_i \tag{3}$$

Eq. 3 thus trivially leads to

$$L_0 \leq L_1 \tag{4}$$

A particular case of Eq. 3 is expressed by Eq. 5 below.

$$C_p < D_I \implies max(D_I, \zeta_i + C_p) < max(D_I, C_p) + \zeta_i \tag{5}$$

Eq. 5 can be read as follows. If the amount of computation is not sufficient to entirely overlap the latency of the inter-node communication ($C_p < D_I$), it is possible to reduce the overall latency of the epoch by additionally overlapping the inter-node communication with the intra-node data transfer. By doing so, the residual overlapping that could not be exploited by the computation only is exploited by the intra-node communication; leading to:

$$C_p < D_I \implies L_0 < L1 \tag{6}$$

If no computation is inserted inside the epoch, then there is no trace of overlapping in any component of Eq. 2; and L_1 becomes:

$$L_{1\_no\_computation} = \alpha_I + \varepsilon_I + D_I + \zeta_i \tag{7}$$

With $C_p = 0$, L_0 is guaranteed to be strictly less than L_1, as shown by the following equation:

$$L_{0\_no\_computation} = \alpha_I + \varepsilon_I + max(D_I, \zeta_i) \tag{8}$$

The currently known message scheduling strive to overlap communication with computation; but they don't push the optimizations to the extent of reaching the latency model expressed by L_0. In particular, MVAPICH and OpenMPI, which are the two major freely available MPI implementations over commodity RDMA-enabled networks do not offer a design compatible with Eq. 1. We offer one such design in a new RMA implementation.

The preceding overlapping analysis assumes that the data transfer is initiated from the origin-side; as it is currently the case in MPICH, MVAPICH and Open MPI. Unless it is justified by zero-copy mechanisms such as CMA [15] or XPMEM [16], It helps to notice that target-initiated RMA transfer schemes lead to additional FIN control messages and are more synchronizing than origin-initiated RMA transfer schemes. Plus, in MPI-3.0 RMA, zero-copy can be achieved without CMA or XPMEM if `MPI_WIN_ALLOCATE` is used; voiding the need for the more cumbersome target-initiated transfer approaches. Finally, by resorting to a dedicated thread, the overlapping issue is trivially solved. However, due to the risks of oversubscription or decreased parallelism, the thread dedication approach is usually not practical in MPI [2, 19].

4. DESIGN EXPLANATION

The design is realized in a completely new RMA-3.0 implementation. For the sake of designation, we will refer to our implementation from now on as NewRMA. We consider a payload threshold th below which an intra-node message is considered too small to be deferred. That threshold is 8KB by default; but it can be modified via an environment variable. We designate respectively by IR, iR and ir an inter-node RMA, an intra-node RMA whose payload is beyond or equal to th and an intra-node RMA whose payload is below th. An RMA communication toward the very issuing origin is treated as any other intra-node RMA. We additionally use the right hand side subscript $_a$ to specify that an RMA can be internally issued because its target has granted access. So IR_a means that the RMA data transfer can be performed while IR means that it cannot. The epoch is separated in the "intra-epoch" phase which precedes the epoch-closing synchronization and the "blocking" phase which occurs inside the epoch-closing synchronization.

In the intra-epoch phase, all the IR_a are issued on the call site. All the ir_a are issued on the call site as well. In fact, it is counter-productive to queue an RMA communication whose payload is 1 byte for instance. In general, it is counterproductive to queue any ready-to-be-transferred intra-node RMA unless its payload is at least th. All the iR_a are queued; even if they are a mere `memcpy` meant for the process itself. In fact, issuing the iR_a on the call site means that its copy cannot be overlapped with any potential inter-node data transfer that comes afterwards. In particular, since large intra-node payloads occupy the CPU for a noticeable amount of time, it is a waste not to overlap that duration with some network activity. This situation is especially important if there is any possibility of any inter-node communication coming after the intra-node RMA. Furthermore, like any other intra-node communication, local data copy inside the intra-epoch phase competes with any computation that could be overlapped inside the epoch. In certain situations, that competition lengthens the time to completion of the epoch. The progress engine activity on RMA call sites is shown in lines 14-30 of Listing 1.

Once the blocking phase is reached (lines 61-66 of Listing 1), the progress engine does several iterations of which each (lines 44-58 of Listing 1) consists of first issuing all the IR_a, then the ir_a and iR_a; and then checking the completion of the all previously issued IR_a The completion checking concerns all the previously issued IR_a; not just those of the current iteration. The inter-node/intra-node communication overlapping occurs because the intra-node data transfer takes place between the initiation and the completion of the IR_a. From a given iteration to another one, some previous IR, iR and ir respectively become IR_a, iR_a and ir_a and the cycles continues until all data transfers complete. We emphasize that no application-level modification is required to benefit from the modifications implemented in NewRMA.

```
 1
 2 /*DATA AND GLOBAL DEFINITIONS*/
 3 th /*threshold beyond which an intra node
     message is deferred*/
 4
 5 MPI_Win_data {
 6 /*Everything in this data stucture is a per
     -window object*/
 7 intra_n_rma_q /*Queue of deferred intra-
     node RMA*/
 8 inter_n_rma_q /*Queue of deferred inter-
     node RMA*/
 9 pending_inter_node_rma_q /*Queue of inter-
     node RMA that are posted and pending
     completion*/
10 }
11 /*END DATA AND GLOBAL DEFINITIONS*/
12
13
14 /*The function below is invoked in the
     intra-epoch phases*/
15 function issue_rma_communication(win, rma){
16   if(is_intra_node(win, rma)){
17     if(target_exposed(win, rma)) && rma->
     data_size < th)
18       issue_intra_node(win, rma)
19     else
20       enqueue(win->intra_n_rma_q, rma)
21   }
22   else {
23     if(target_exposed(win, rma)){
24       issue_inter_node(win, rma)
25       enqueue(win->pending_inter_node_rma_q
     , rma)
26     }
27     else
28       enqueue(win->inter_n_rma_q, rma)
29   }
30   iterate_on_inter_node_rma()
31 }
32
33
34 function iterate_on_inter_node_rma(){
35   foreach(win in all_active_rma_windows){
36     foreach(rma in win->inter_n_rma_q){
37       if(target_exposed(win, rma)){
38         issue_inter_node(win, rma)
39         enqueue(win->
     pending_inter_node_rma_q, rma)
40         dequeue(win->inter_n_rma_q, rma)
41       }
42     }
43   }
44 }
```

```
function iterate_on_progression(){
  iterate_on_inter_node_rma()
  foreach(win in all_active_rma_windows){
    foreach(rma in win->intra_n_rma_q){
      if(target_exposed(win, rma)){
        issue_intra_node(win, rma)
        dequeue(win->intra_n_rma_q, rma)
      }
    }
    foreach(rma in win->
    pending_inter_node_rma_q){
      if(has_completed(rma))
        dequeue(win->
    pending_inter_node_rma_q, rma)
    }
  }
}

//blocking phase
function close_epoch(win){
  ...
  while(has_pending_communications(win) ||
    has_queued_rma(win))
    iterate_on_progression()
  ...
}
```

Listing 1: Progress engine pseudo-code

5. EVALUATION

On an QDR Mellanox ConnectX InfiniBand cluster made of 32 nodes, each having two quad-core Nehalem x5550 CPU of 2.6 GHz and 36 GB of memory per node, we show an example of comparison between how our implementation of Radix performs when executed with NewRMA or with the vanilla MVAPICH 2-1.9. We sort 2^{29}, 2^{31} and 2^{33} 64-bit integers generated uniformly. Each test result, shown in Fig. 1, is the average over 4 iterations. While NewRMA consistently outperforms MVAPICH, it is not possible to isolate how much of the performance difference is due to the message scheduling scheme being discussed in the current work. In fact NewRMA differs from the vanilla MVAPICH 2-1.9 RMA in many respects. The isolation is nevertheless possible with the carefully crafted microbenchmark tests discussed below.

5.1 Experimental Evaluation of the Message Scheduling Effect

The experimental framework is a four node InfiniBand cluster. Each node is a dual-socket quad core 2GHz AMD

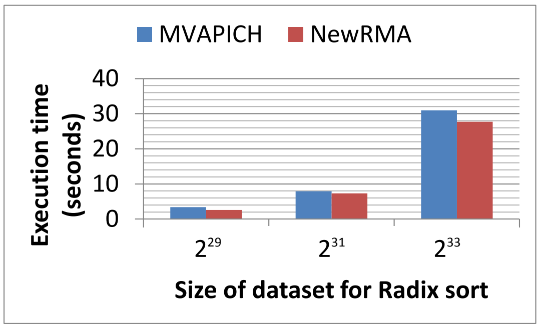


Figure 1: Example of comparative MVAPICH/NewRMA performance of Radix sort over 256 CPU cores

Opteron 2350 CPU with 8GB of RAM. The network devices are Mellanox ConnectX QDR HCA and switches. The tests are performed with the vanilla MVAPICH 2-1.9, Open MPI 1.8.1 and our NewRMA implemented inside MVAPICH 2-1.9. Each result is the average over 100 iterations. All the tests are performed with a single origin and a limited number of targets so as to 1) avoid the effects of network device-level congestion and 2) clearly make the overlapping effects observable at a glance on the figures.

The single origin process performs RMA communications toward a certain number of targets. In a first set of tests shown in Fig. 2, there is no RMA from the origin process toward itself. In a second set of tests shown in Fig. 3, the origin performs RMA toward itself. We compare the following scenarios:

1. The origin performs intra-node communications with three targets on the same node. In the tests related to Fig. 2, a total of four processes is used as the origin is not allowed to communicate with itself. In the tests related to Fig. 3, three processes are used since the origin is target as well. The test series for this scenario is named `Intra` in Fig. 2 and Fig. 3.

2. The origin performs inter-node communications. A total of four processes is used; that is, the origin plus three targets on three distinct nodes. The test series for this scenario is named `Inter` in Fig. 2 and Fig. 3.

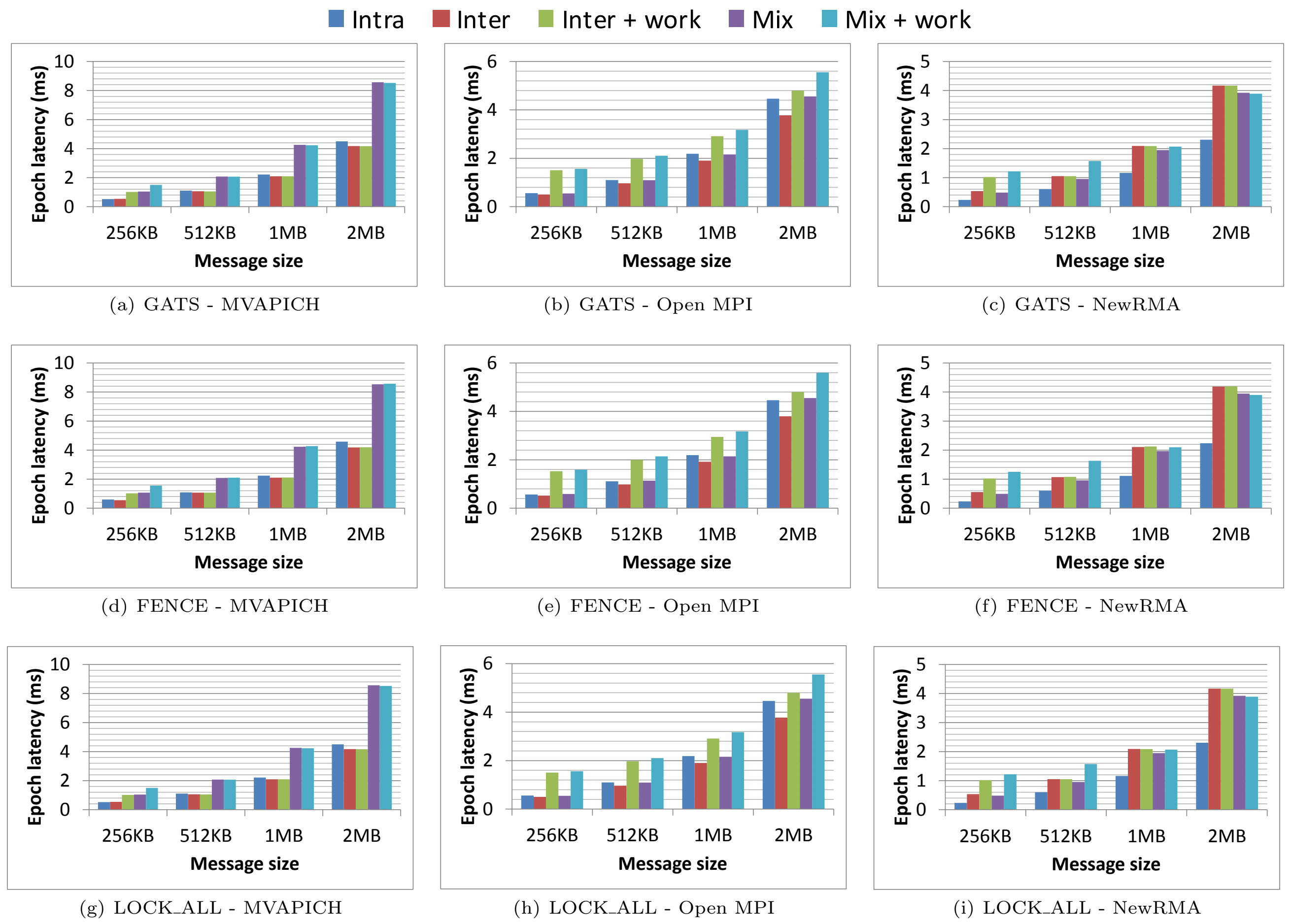
3. The origin performs inter-node communications only but overlaps 1000μs of work inside the epoch. The test series for this scenario is named `Inter+work` in Fig. 2 and Fig. 3. One can notice that it is not useful to reproduce the equivalent of this scenario for intra-node communications because, only one *engine*, the CPU, is available in the intra-node-only test; there is no RDMA.

4. The origin performs a mix of intra-node and inter-node communications. In Fig. 2, seven processes are used while six are used for Fig. 3. The test series for this scenario is named `Mix` in Fig. 2 and Fig. 3.

5. The origin performs a mix of inter-node and intra-node communications; and works 1000μs inside the epoch. The test series for this scenario is named `Mix+work` in Fig. 2 and Fig. 3.

The tests are performed for three epoch types, namely, GATS, fence and lock_all. The passive target test cannot simply be based on `MPI_WIN_LOCK`; it must be based on `MPI_WIN_LOCK_ALL` to allow the origin to communicate with multiple peers in the same epoch. The tests are performed for `MPI_PUT` and `MPI_GET`. However, the observations are identical for both kinds of RMA operations. Consequently, the analysis is performed only with `MPI_PUT`.

In Fig. 2, the `Intra` test series shows that it takes about 4.5μs to transfer 2MB for MVAPICH (Fig. 2(a), Fig. 2(d), Fig. 2(g)) and for Open MPI (Fig. 2(b), Fig. 2(e), Fig. 2(h)) while it only takes about 2.3μs to transfer the same amount of data with NewRMA (Fig. 2(c), Fig. 2(f), Fig. 2(i)). The reason for that difference is that NewRMA does zero intermediate shared memory copy for its intra-node data transfers. The difference just shown is only one of many that exist between the three implementations. By putting each



(a) GATS - MVAPICH (b) GATS - Open MPI (c) GATS - NewRMA
(d) FENCE - MVAPICH (e) FENCE - Open MPI (f) FENCE - NewRMA
(g) LOCK_ALL - MVAPICH (h) LOCK_ALL - Open MPI (i) LOCK_ALL - NewRMA

Figure 2: Overlapping behaviour comparisons. Origin is not target

implementation on a separate graph, we seek to isolate the various design or implementation differences that might justify the various levels of performance gaps between them. Each implementation is then compared against itself for the analysis that we build over the five test series.

As a reminder, for two activities of duration C_p and D, full overlapping occurs if the aggregated length l of both activities is $l = max(D, C_p)$. No overlapping occurs if $l = D + C_p$; and partial overlapping occurs if $max(D, C_p) < l < D + C_p$.

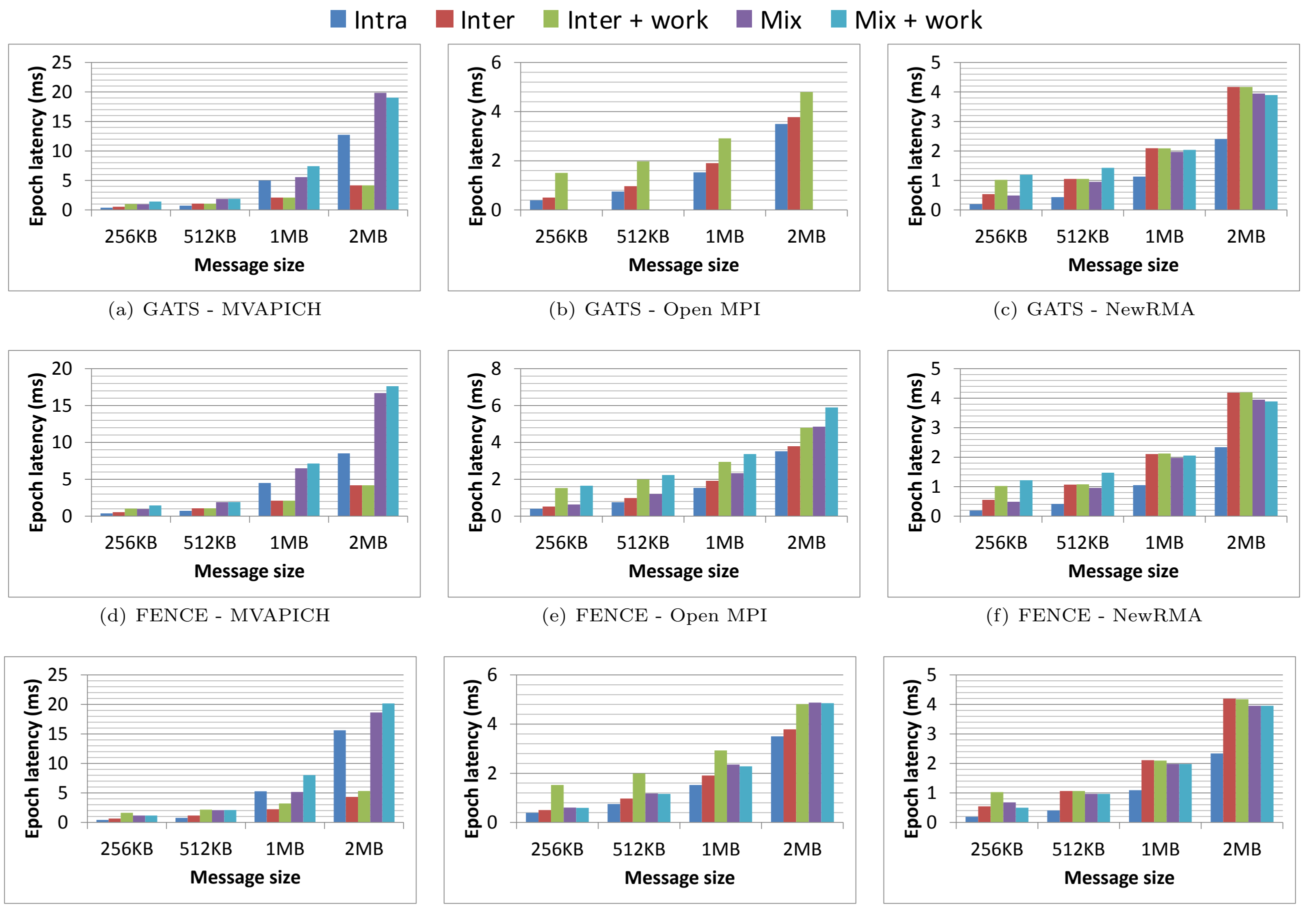
The analysis starts with the case of the origin not issuing RMA toward itself (Fig. 2). For all three epochs, when we observe `Inter` and `Inter+work`, we can notice that MVAPICH (Fig. 2(a), Fig. 2(d), Fig. 2(g)) and NewRMA (Fig. 2(c), Fig. 2(f), Fig. 2(i)) offer communication/computation overlapping because they show the pattern $l = max(D, C_p)$. Open MPI (Fig. 2(b), Fig. 2(e), Fig. 2(h)) does not offer communication/computation overlapping. From the epoch latency point of view, MVAPICH and NewRMA behave according to Eq. 2; with ζ_i being zero in this case.

Still in Fig. 2, when we compare `Mix` with `Intra` and `Inter`, we notice for MVAPICH (Fig. 2(a), Fig. 2(d), Fig. 2(g)) that `Mix` is (at least) the sum of `Intra` and Inter. For showing the pattern $l = D + C_p$, MVAPICH does not achieve any overlapping between intra-node and inter-node communication. In comparison, Open MPI (Fig. 2(b), Fig. 2(e), Fig. 2(h)) and NewRMA (Fig. 2(c), Fig. 2(f), Fig. 2(i)) show behaviours that express intra-node/inter-node communication overlapping as the `Mix` test series is equivalent to one of `Inter` or `Intra` (pattern $l = max(D, C_p)$ and Eq. 8).

Finally, when considering the `Mix+work` test series, we can observe that NewRMA does better than both MVAPICH and Open MPI. First, the lack of communication/computation overlapping in Open MPI makes its `Mix+work` test series more expensive than its `Mix` test results (Fig. 2(b), Fig. 2(e), Fig. 2(h)). While the `Mix+work` test series of MVAPICH is just as expensive as its Mix test series, one should remember that the `Mix` test series of MVAPICH was already deprived of intra-node/inter-node communication overlapping (Fig. 2(a), Fig. 2(d), Fig. 2(g)). The `Mix+work` case of MVAPICH behaves according to Eq. 2. NewRMA (Fig. 2(c), Fig. 2(f), Fig. 2(i)) effectively exploits any residual overlapping potential that is not exploited by the computation by parallelizing the intra-node data copy with the inter-node data transfer. The behaviour of NewRMA is according to Eq. 1. In fact, one can notice that Mix + work is not more costly than Mix; but more importantly, it is not even more costly than `Inter`.

Fig. 2(c), Fig. 2(f) and Fig. 2(i) show that `Mix` and Mix + work tend to slightly outperform `Inter` and `Inter+work` respectively. We hypothesize that this observation is linked to cache behaviour.

When the origin issues RMA communications toward itself (Fig. 3), all the previous analysis hold again. We have



(a) GATS - MVAPICH (b) GATS - Open MPI (c) GATS - NewRMA

(d) FENCE - MVAPICH (e) FENCE - Open MPI (f) FENCE - NewRMA

(g) LOCK_ALL - MVAPICH (h) LOCK_ALL - Open MPI (i) LOCK_ALL - NewRMA

Figure 3: Overlapping behaviour comparisons. Origin is target too

not been able to issue RMA communications to self in Open MPI when the origin performs both intra-node and inter-node communications and communicates with itself. As a result, Fig. 3(b) is incomplete. In the other epochs, Open MPI (Fig. 3(e), Fig. 3(h)) no longer offers full intra-node/inter-node communication overlapping as the `Mix` results are noticeably more expensive than the maximum of `Intra` and `Inter`. This change is due to Open MPI doing right on the call site the data copy of RMA communications toward the process itself. Consequently, any inter-node RMA that is issued after the RMA-to-self cannot be overlapped with the copy of that RMA-to-self. In comparison, NewRMA (Fig. 3(c), Fig. 3(f), Fig. 3(i)) maintains all the positive behaviours that it showed in the case of the origin not communicating with itself.

6. RELATED WORK

Overlapping in MPI has been mostly about parallelizing communication and computation. The literature is particularly abundant for communication/computation overlapping for two-sided communications [8, 12, 13, 19]. In order to mitigate the synchronization effects, previous MPI one-sided communication designs deferred the blocking steps to the epoch-closing routine [1, 14]. This approach, termed *lazy* has the major drawback of annihilating any communication/computation overlapping potential. In MPI one-sided communications, communication/computation overlapping efforts have leveraged the use of RDMA to move away from the early two-sided communication-based design of MPI RMA [3, 5, 11]. In [10] a design of the fence epoch is proposed where communication/computation overlapping occurs inside the epoch. However, since the proposal makes every single fence call blocking, the overlapping comes at the price of a potentially substantial idleness at both opening and closing of each epoch. In [18], a strategy is proposed to adaptively switch between lazy and eager modes for RMA communications to achieve overlapping.

Purely intra-node RMA issues have been addressed in [4, 6, 9]. However, To the best of our knowledge, message scheduling in MPI one-sided communication in order to reduce the overall epoch latency has not been attempted before.

7. CONCLUSION

We proposed an intra-epoch message scheduling scheme that overlaps intra-node communication with inter-node communication. In situations where no computation is available to overlap with the inter-node data transfer, our message scheduling scheme exploits the unused overlapping potential to reduce the overall latency of the RMA epoch. The same effect is observed, but to a lesser extent, when computation can be overlapped with the RMA communication; but the amount of communication is not sufficient to exploit the whole overlapping potential. Comparisons with MVAPICH and Open MPI show the benefits provided by the proposed message scheduling scheme.

As future work, we intend to incorporate in the scheduling algorithm the bandwidth information of the system; as well as the average latency of `memcpy` for ranges of payload sizes. Once the epoch is closed, those information would help auto-tune the threshold that determine if an intra-node RMA transfer is deferred or issued immediately.

8. ACKNOWLEDGEMENTS

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada Grant #RGPIN/238964-2011, Canada Foundation for Innovation and Ontario Innovation Trust Grant #7154.

We thank the Argonne Leadership Computing Facility for the Fusion cluster as well as Mellanox Technologies for the resources used to conduct this study.

9. REFERENCES

[1] B. Barrett, G. Shipman, and A. Lumsdaine. Analysis of Implementation Options for MPI-2 One-Sided. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 4757 of *Lecture Notes in Computer Science*, pages 242–250. Springer Berlin Heidelberg, 2007.

[2] T. Hoefler and A. Lumsdaine. Message Progression in Parallel Computing - to Thread or Not to Thread? In *Proceedings of the 2008 IEEE International Conference on Cluster Computing (Cluster)*, pages 213–222, 2008.

[3] W. Jiang, J. Liu, H.-W. Jin, D. Panda, W. Gropp, and R. Thakur. High Performance MPI-2 One-Sided Communication over InfiniBand. In *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 531–538, 2004.

[4] P. Lai, S. Sur, and D. Panda. Designing Truly One-sided MPI-2 RMA Intra-node Communication on Multi-core Systems. *Computer Science - Research and Development*, 25(1-2):3–14, 2010.

[5] J. Liu, W. Jiang, P. Wyckoff, D. K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen. Design and Implementation of MPICH2 over InfiniBand with RDMA Support. In *Proceedings of the 2004 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, 2004.

[6] M. Luo, S. Potluri, P. Lai, E. Mancini, H. Subramoni, K. Kandalla, S. Sur, and D. Panda. High Performance Design and Implementation of Nemesis Communication Layer for Two-Sided and One-Sided MPI Semantics in MVAPICH2. In *Proceedings of the 2010 International Conference on Parallel Processing Workshops (ICPPW)*, pages 377–386, 2010.

[7] MPI Forum. The Message Passing Interface. `http://www.mpi-forum.org/`. Online; accessed 2014-05-11.

[8] M. J. Rashti and A. Afsahi. A Speculative and Adaptive MPI Rendezvous Protocol Over RDMA-enabled Interconnects. *International Journal of Parallel Programming*, 37(2):223–246, 2009.

[9] G. Santhanaraman, P. Balaji, K. Gopalakrishnan, R. Thakur, W. Gropp, and D. K. Panda. Natively Supporting True One-Sided Communication in MPI on Multi-core Systems with InfiniBand. In *Proceedings of the 2009 IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 380–387. IEEE Computer Society, 2009.

[10] G. Santhanaraman, T. Gangadharappa, S. Narravula, A. Mamidala, and D. Panda. Design Alternatives for Implementing Fence Synchronization in MPI-2 One-sided Communication for InfiniBand Clusters. In *Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops (Cluster)*, pages 1–9, 2009.

[11] G. Santhanaraman, S. Narravula, and D. Panda. Designing Passive Synchronization for MPI-2 One-sided Communication to Maximize Overlap. In *Proceedings of the 2008 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–11, 2008.

[12] M. Small and X. Yuan. Maximizing MPI Point-to-point Communication Performance on RDMA-enabled Clusters with Customized Protocols. In *Proceedings of the 2009 International Conference on Supercomputing (ICS)*, pages 306–315. ACM, 2009.

[13] S. Sur, H.-W. Jin, L. Chai, and D. K. Panda. RDMA Read Based Rendezvous Protocol for MPI over InfiniBand: Design Alternatives and Benefits. In *Proceedings of the 2006 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 32–39. ACM, 2006.

[14] R. Thakur, W. Gropp, and B. Toonen. Optimizing the Synchronization Operations in Message Passing Interface One-Sided Communication. *International Journal of High Performance Computing Application (IJHPCA)*, 19:119–128, 2005.

[15] J. Vienne. Benefits of Cross Memory Attach for MPI Libraries on HPC Clusters. In *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*, XSEDE '14, pages 33:1–33:6, New York, NY, USA, 2014. ACM.

[16] M. Woodacre, D. Robb, D. Roe, and K. Feind. The SGI Altix 3000 Global Shared-Memory Architecture.

[17] K. Zhang and B. Wu. A novel parallel approach of radix sort with bucket partition preprocess. In *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pages 989–994, June 2012.

[18] X. Zhao, G. Santhanaraman, and W. Gropp. Adaptive Strategy for One-Sided Communication in MPICH2. In *Recent Advances in the Message Passing Interface*, volume 7490 of *Lecture Notes in Computer Science*, pages 16–26. Springer Berlin Heidelberg, 2012.

[19] J. A. Zounmevo and A. Afsahi. Investigating Scenario-Conscious Asynchronous Rendezvous over RDMA. In *Proceedings of the 2011 IEEE International Conference on Cluster Computing (Cluster)*, pages 542–546, 2011.